যে কোন প্রোগ্রাম লিখতে গেলেই আসলে আমাদের আই/ও দরকার হয়। তবে এই বিষয়টি ঠিক ততটা মজার না যতটা অন্যান্য বিষয় গুলো। খানিকটা ইলেক্ট্রিসিটিরর মতো। আমরা জানি প্রত্যেকটি বাড়িতেই এটি আছে, দরজা দিয়ে প্রবেশ করেই আমাদের হাত সুইচবোর্ডের দিকে চলে যায়, আমার সুইচ টিপ দিই, এবং লাইট জ্বলে উঠে। এর পেছনের ব্যপারগুলো নিয়ে যেমন ইলেক্ট্রিসিটি কোথা থেকে এলো, কিভাবে কাজ করে এসব নিয়ে আমাদের চিন্তা করতে হয় না। এগুলো নেপথ্যে থেকে ঠিক ঠাক মতো কাজ করে। আই/ও অনেকটা এরকম।

একটি প্রোগ্রাম মূলত ডাটা আর ফাংশন এর সমষ্টি। অর্থাৎ ফাংশন ডাটা গুলো নিয়ে কাজ করে। তো এই ডাটা গুলো কোথাও থেকে তৈরি হয় এবং সেগুলোকে আমাদের প্রোগ্রাম ফাংশন প্রসেস করে । প্রসেসকৃত ডাটা গুলো হচ্ছে আউটপুট। সহজ করে বলা যেতে পারে, আমাদের প্রোগ্রাম কোন সোর্স থেকে ডাটা পড়ে এবং কোন একটা ডেস্টিনেশনে রাইট করে। উদাহরণ হিসেবে দেওয়া যেতে পারে- আমাদের কিবোর্ড একটি ডাটা সোর্স। আমরা একটা প্রোগ্রাম লিখতে পারি যা কি বোর্ড এ ডাটা টাইপ করছি তা ইনপুট হিসেবে নিচ্ছে এবং System.out.println() মেথড দিয়ে সেগুলো কনসোলে প্রিন্ট করতে পারি।

তবে একজন জাভা প্রোগ্রামার এর কাছে আই/ও অনেক গুলো কারণেই গুরুত্বপূর্ণ হতে পারে। জাভাতে অনেক গুলো আই/ও ক্লাস এর কোর এপিআই এর সাথেই থাকে যার বেশির ভাগ – java.io প্যাকেজ-এ। তবে জাভাতে অধিকাংশ ক্ষেত্রেই আই/ও দুই ভাগে ভাগ করা হয়েছে। একটি হলো বাইট ভিত্তিক আই/ও যা input stream এবং output stream দিয়ে হ্যান্ডেল করা হয়, এবং অন্যটি হলো ক্যারেকটার ভিত্তিক যা readers এবং writers দিয়ে হ্যান্ডেল করা হয়। তবে দুই টাইপ-এ অ্যাবস্ট্রাকশন সরবরাহ করে যা দিয়ে সোর্সের সঠিক টাইপ না জেনেও পড়তে বা লিখতে পারি। এতে করে আমরা একি মেথড দিয়ে কনসোল থেকে ডাটা পড়তে পারছি আবার সেই মেথড দিয়ে আমরা নেটওয়ার্ক কানেকশন থেকেও পড়তে পারছি। এতো হল টিপ অব দি আইসবার্গ। একবার আমরা অ্যাবস্ট্রাকশন এ অভ্যস্ত হয়ে গেলে যে কোন সোর্স থেকে ডাটা পড়তে পারবো, আমাদের আসলে খুব একটা কেয়ার করতে হবে না কিভাবে বা কোন সোর্স থেকে ডাটা আসছে বা যাচ্ছে। এখানে একটা গুরুত্বপূর্ণ কথা বলে রাখি, সেটা হলো, জাভা প্রোগ্রামারদের সব থেকে পছন্দের বিষয় হচ্ছে অ্যাবস্ট্রাকশন। অনেক ভূমিকা হলো, এবার তাহলে আরো ভেতরে প্রবেশ করা যাক। শুরতেই ফাইল নিয়ে কাজ করা যাক।

**ওয়ার্কিং উয়িদ ফাইল**

**পাথ** প্রত্যেকটি ফাইল এর জন্যে একটি নির্দিষ্ট পাথ থাকে যাতে করে আমরা আলাদা করতে পারি। পাথ হচ্ছে কতগুলো ক্যারেকটার এর সমষ্টি এবং এতে ফাইলে এর নাম এবং ডিরেকটরী লোকাশান থাকে।

যেমন ওয়িন্ডোস প্লাটফর্মের ক্ষেত্রে C:\users\rokonoid\hello.txt হচ্ছে hello.txt ফাইল এর পাথনেইম যা কিনা C ড্রাইভের users ডিরেকটরির মাঝে rokonoid ডিরেকটরিতে আছে। Unix প্লাটফর্মের ক্ষেত্রে /home/rokonoid/hello.txt হচ্ছে hello.txt এর পাথনেইম।

পাথনেইম দুই প্রকার হতে পারে- absolute path এবং relative path. Current working directory বলে একটা কনসেপ্ট আছে, আর সেটি হলো, আমরা যখন যে ডিরেকটরিতে কাজ করি। মনে করা যাক আমাদের জাভা প্রোগ্রামটি /home/rokonoid বা C:\users\rokonoid ডিরেকটরিতে আছে । তাহলে আমাদের কারেন্ট ওয়ার্কিং ডিরেকটরি হচ্ছে C:\users\rokonoid বা /home/rokonoid। এখন এই ডিরেকটরিতে যদি একটি hello.txt ফাইল থাকে, তাহলে এই ফাইল এর রিলেটিভ পাথ হবে hello.txt আর absolute path পাথ হবে C:\users\rokonoid\hello.txt বা /home/rokonoid/hello.txt । রিলেটিভ পাথ কারেন্ট ওয়ার্কিক ডিরেকটরি থেকে রিজলভ করা যায়।

**ফাইল তৈরি**

এবার দেখা যাক কিভাবে একটি ফাইল অবজেক্ট তৈরি করা যায়। *java.io.File* ক্লাসটি একটি পাথ এর ফাইল বা ডিরেকটরিকে রিপ্রেজেন্ট করে। এ ক্লাসে বেশ কয়েকটি কনস্ট্রাকর রয়েছে,এর মানে বেশ কয়েক উপায়ে একটি ফাইল অবজেক্ট তৈরি করা যায়।

File(String pathname)

File(File parent, String child)

File(String parent, String child)

File(URI uri)

এখন আমাদের একটি পাথনেইম যদি হয় hello.txt বা /home/rokonoid/hello.txt তাহলে আমরা নিচের মতো করে ফাইল অবজেক্ট তৈরি করতে পারি।

File file = new File("hello.txt");

অথবা

File file = new File("/home/rokonoid/hello.txt");

এই ফাইলটি আমাদের দেওয়া পাথ এ যে ফাইলটি আছে তাকে রিপ্রেজেন্ট করে। তবে মজার ব্যপার হচ্ছে ফাইল অবজেক্ট তৈরি করতে হলে এই পাথটি ফিজিক্যালি থাকতে হবে এমন কোন কথা নেই। File ক্লাসের বেশি কিছু মেথড আছে যেগুলো দিয়ে আমরা দেখতে পারি এই পাইলটি আসলেই আমরা যে পাথটি দিয়েছি সেখানে আছি কিনা। না থাকলে আমরা তৈরি করতে পারি।

import java.io.File;

import java.io.IOException;

public class FileExample {

public static void main(String[] args) {

File file = new File("hello.txt");

if (file.exists()) {

System.out.println("File exists");

} else {

System.out.println("File does not exist,lets create one");

try {

file.createNewFile();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

উপরের উদহরণটিতে আমরা প্রথমে আমাদের দেওয়া পাথ দিয়ে একটি ফাইল অবজেক্ট তৈরি করেছি। তারপর দেখেছি এই ফাইটি আসলেই ফিজিক্যালি আমাদের দেওয়া পাথ এ আছে কিনা। যদি না থাকে, তাহলে সেই পাথ এ নতুন একটি ফাইল তৈরি করা হয়েছে।

এছাড়াও আরও কিছু বেশ প্রয়োজনীয় মেথড যেমন- isFile() এবং isDirectory() আছে যেগুলো দিয়ে আমরা বের করতে পারি কোন পাথ ফাইল বা ডিরেকটরি কিনা।

এছাড়াও কারেন্ট ওয়ার্কিং ডিরেকটি বের করা জন্যে একটি বিশেষ উপায় হলো -

public class CurrentWorkingDirectory {

public static void main(String[] args) {

String workingDir = System.getProperty("user.dir");

System.out.println(workingDir);

}

}

**পাথ সেপারেটর**

একটি বিষয় মনে রাখতে হবে যে বিভিন্ন প্লাটপর্ম ফাইলের পাথ এর দুটি পার্ট আলাদা করার জন্যে আলাদা ক্যারেকটার ব্যবহার করে থাকে। যেমন- windows ব্যাকস্লেস () এবং unix সিস্টেম ফরওয়ার্ড স্লেস (/) ব্যবহার করে থাকে। সুতরাং পাথ তৈরি করতে হলে খেয়াল রাখা জরুরী কোন প্লাটফর্মে থেকে প্রোগ্রামটি রান করা হচ্ছে। কিন্তু আমাদের যেহেতু মূল উদ্দেশ্য প্লাটপর্ম স্পেসিফিক কোড না লেখা, সেক্ষত্রে নিজের উপায়টি ব্যবহার করা যেতে পারে।

String workingDir = System.getProperty("user.dir");

String newFile = workingDir + File.separator + "hellword.txt";

File file = new File(newFile);

এখানে File.separator একটি কনস্ট্যান্ট যা যে প্লাটফর্মে প্রোগ্রামটি রান করছে তার উপর ভিত্তি করে সেপারেটর স্ট্রিং আকারে দিয়ে থাকে।

**ডিরেকটরি তৈরি**

File ক্লাসে এ mkdir() এবং mkdirs() দুটি মেথড আছে যেগুলো ব্যবহার করে আমরা একটি ডিরেকটরি তৈরি করতে পারি। এবং এদের মাঝে ফাইল তৈরি করতে পারি।

import java.io.File;

import java.io.IOException;

public class DirectoryExample {

public static void main(String[] args) throws IOException {

File dir = new File("/home/rokonoid/myDir");

dir.mkdir();

String dirPath = dir.getPath();

System.out.println("Diectory Path: " + dirPath);

// lets create a new file

String fileName = "hello.txt";

File file = new File(dirPath + File.separator + fileName);

file.createNewFile();

String filePath = file.getPath();

System.out.println("File Path: "+ filePath);

}

}

এই প্রোগ্রামটি রান করলে নিচের আউটপুট পাওয়া যাবে -

Diectory Path: /home/rokonoid/myDir

File Path: /home/rokonoid/myDir/hello.txt

**ফাইল রিনেমিং , কপিং এবং ডিলেটিং**

File ক্লাস এ renameTo() ব্যবহার করে আমরা ফাইল রিনেম করতে পারি।

import java.io.File;

public class FileRenameExample {

public static void main(String[] args) {

File oldFile = new File("old\_hello.txt");

File newFile = new File("new\_hello.txt");

boolean fileRenamed = oldFile.renameTo(newFile);

if (fileRenamed) {

System.out.println(oldFile + " renamed to " + newFile);

} else {

System.out.println("Renaming " + oldFile + " to " + newFile + " failed.");

}

}

}

ফাইল ডিলিট করার জন্যে দুটি মেথড রয়েছে- delete() এবং deleteOnExit() এই মেথড দুটি দিয়ে ফাইল এবং ডিরেকটররী ডিলেট করা যায়। তবে ডিরেকটরী ডিলিট করতে হলে অবশ্যই ডিরেকটরি টি খালি থাকতে হবে, অর্থাৎ ডিরেকটরীতে যদি আরও ফাইল থাকে , তাহলে সেগুলো আগে ডিলিট করে ফেলতে হবে। delete() মেথডটি সাথে সাথেই কাজ করে তবে, deleteOnExit() মেথডটি যখন JVM টারমিনেট করে তখন ডিলেট করে। আমাদের অনেকসময় প্রোগ্রাম চলাকালিন টেম্পোরারি ফাইল তৈরি করার দরকার পরে যা প্রোগ্রাম টার্মিনেট হয়ে গেলে দরকার হয় না,সেসব ক্ষেত্রে এই মেথড ব্যবহার করা যেতে পারে।

public class FileDeleteExample {

public static void main(String[] args) {

// To delete the hello.txt file immediately

File file1 = new File("hello.txt");

file1.delete();

// To delete the hello.txt file when the JVM terminates

File file2 = new File("hello.txt");

file2.deleteOnExit();

}

}

File ক্লাসে কোন মেথড নেই যাতে করে সরাসরি আমরা ফাইল কপি করতে পারি। একটি ফাইল কপি করতে হলে আমাদেরকে একটি নতুন ফাইল তৈরি করতে হবে এবং সেই ফাইল এর কন্টেন্ট গুলো রিড করে নতুন ফাইল এ রাইট করতে হবে। পরবর্তি চ্যাপ্টারে এ নিয়ে আলোচনা করা হবে। লিস্টিং ফাইলস

আমরা একটি ডিরেকটরিতে কতগুলো ফাইল আছে তার লিস্ট listFiles() মেথড দিয়ে সহজেই বের করে ফেলতে পারি। উদাহরণ-

import java.io.File;

public class ListingFiles {

public static void main(String[] args) {

File home = new File("/home/rokonoid/");

File[] listRoots = home.listFiles();

for (File file : listRoots) {

System.out.println(file.getPath());

}

}

}

**ফাইল ফিল্টার**

তবে অনেক সময় আমাদের ফাইল ফিল্টারের প্রয়োজন হয়। মনে করা যাক একটি ডিরেকটরীতে শুধুমাত্র png ফাইল গুলো আমাদের দরকার। সেক্ষেত্রে -

import java.io.File;

import java.io.FileFilter;

public class FileFileterExample {

public static void main(String[] args) {

File home = new File("/home/rokonoid/Pictures");

FileFilter pngFlter = new FileFilter() {

@Override

public boolean accept(File pathname) {

String fileName = pathname.getName();

if (fileName.endsWith(".png")) {

return true;

}

return false;

}

};

File[] listRoots = home.listFiles(pngFlter);

for (File file : listRoots) {

System.out.println(file.getPath());

}

}

}

উপরের উদাহরণটিতে FileFilter এর একটি anonymous ক্লাস লেখা হয়েছে যা কিনা listFiles() মেথডটি paremeter হিসেবে নিচ্ছে । এই ফিল্টারের accept() মেথডটিতে আমরা আমাদের ফিল্টার লজিকটুকু লেখা হয়েছে যাতে করে এটি শুধুামাত্র png ফাইল গুলো লিস্টিং করে।

**ডিরেকটরি তৈরি**

File ক্লাসে এ mkdir() এবং mkdirs() দুটি মেথড আছে যেগুলো ব্যবহার করে আমরা একটি ডিরেকটরি তৈরি করতে পারি। এবং এদের মাঝে ফাইল তৈরি করতে পারি।

import java.io.File;

import java.io.IOException;

public class DirectoryExample {

public static void main(String[] args) throws IOException {

File dir = new File("/home/rokonoid/myDir");

dir.mkdir();

String dirPath = dir.getPath();

System.out.println("Diectory Path: " + dirPath);

// lets create a new file

String fileName = "hello.txt";

File file = new File(dirPath + File.separator + fileName);

file.createNewFile();

String filePath = file.getPath();

System.out.println("File Path: "+ filePath);

}

}

এই প্রোগ্রামটি রান করলে নিচের আউটপুট পাওয়া যাবে -

Diectory Path: /home/rokonoid/myDir

File Path: /home/rokonoid/myDir/hello.txt

**ফাইল রিনেমিং , কপিং এবং ডিলেটিং**

File ক্লাস এ renameTo() ব্যবহার করে আমরা ফাইল রিনেম করতে পারি।

import java.io.File;

public class FileRenameExample {

public static void main(String[] args) {

File oldFile = new File("old\_hello.txt");

File newFile = new File("new\_hello.txt");

boolean fileRenamed = oldFile.renameTo(newFile);

if (fileRenamed) {

System.out.println(oldFile + " renamed to " + newFile);

} else {

System.out.println("Renaming " + oldFile + " to " + newFile + " failed.");

}

}

}

ফাইল ডিলিট করার জন্যে দুটি মেথড রয়েছে- delete() এবং deleteOnExit() এই মেথড দুটি দিয়ে ফাইল এবং ডিরেকটররী ডিলেট করা যায়। তবে ডিরেকটরী ডিলিট করতে হলে অবশ্যই ডিরেকটরি টি খালি থাকতে হবে, অর্থাৎ ডিরেকটরীতে যদি আরও ফাইল থাকে , তাহলে সেগুলো আগে ডিলিট করে ফেলতে হবে। delete() মেথডটি সাথে সাথেই কাজ করে তবে, deleteOnExit() মেথডটি যখন JVM টারমিনেট করে তখন ডিলেট করে। আমাদের অনেকসময় প্রোগ্রাম চলাকালিন টেম্পোরারি ফাইল তৈরি করার দরকার পরে যা প্রোগ্রাম টার্মিনেট হয়ে গেলে দরকার হয় না,সেসব ক্ষেত্রে এই মেথড ব্যবহার করা যেতে পারে।

public class FileDeleteExample {

public static void main(String[] args) {

// To delete the hello.txt file immediately

File file1 = new File("hello.txt");

file1.delete();

// To delete the hello.txt file when the JVM terminates

File file2 = new File("hello.txt");

file2.deleteOnExit();

}

}

File ক্লাসে কোন মেথড নেই যাতে করে সরাসরি আমরা ফাইল কপি করতে পারি। একটি ফাইল কপি করতে হলে আমাদেরকে একটি নতুন ফাইল তৈরি করতে হবে এবং সেই ফাইল এর কন্টেন্ট গুলো রিড করে নতুন ফাইল এ রাইট করতে হবে। পরবর্তি চ্যাপ্টারে এ নিয়ে আলোচনা করা হবে। লিস্টিং ফাইলস

আমরা একটি ডিরেকটরিতে কতগুলো ফাইল আছে তার লিস্ট listFiles() মেথড দিয়ে সহজেই বের করে ফেলতে পারি। উদাহরণ-

import java.io.File;

public class ListingFiles {

public static void main(String[] args) {

File home = new File("/home/rokonoid/");

File[] listRoots = home.listFiles();

for (File file : listRoots) {

System.out.println(file.getPath());

}

}

}

**ফাইল ফিল্টার**

তবে অনেক সময় আমাদের ফাইল ফিল্টারের প্রয়োজন হয়। মনে করা যাক একটি ডিরেকটরীতে শুধুমাত্র png ফাইল গুলো আমাদের দরকার। সেক্ষেত্রে -

import java.io.File;

import java.io.FileFilter;

public class FileFileterExample {

public static void main(String[] args) {

File home = new File("/home/rokonoid/Pictures");

FileFilter pngFlter = new FileFilter() {

@Override

public boolean accept(File pathname) {

String fileName = pathname.getName();

if (fileName.endsWith(".png")) {

return true;

}

return false;

}

};

File[] listRoots = home.listFiles(pngFlter);

for (File file : listRoots) {

System.out.println(file.getPath());

}

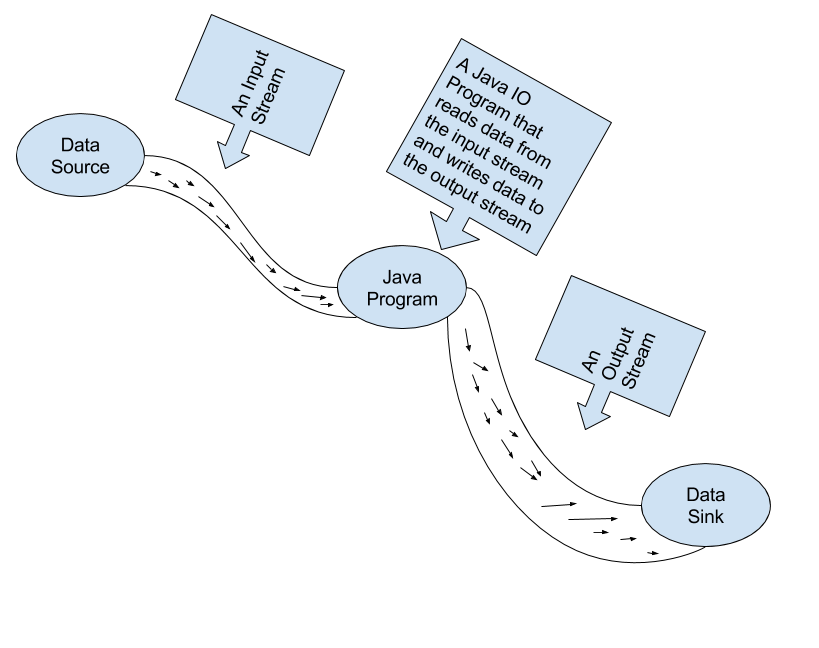
}

}

উপরের উদাহরণটিতে FileFilter এর একটি anonymous ক্লাস লেখা হয়েছে যা কিনা listFiles() মেথডটি paremeter হিসেবে নিচ্ছে । এই ফিল্টারের accept() মেথডটিতে আমরা আমাদের ফিল্টার লজিকটুকু লেখা হয়েছে যাতে করে এটি শুধুামাত্র png ফাইল গুলো লিস্টিং করে।

**ইনপুুট/আউটপুট স্ট্রিম**

স্ট্রিম এরে আক্ষরিক অর্থ হচ্ছে প্রবাহ । এর মানে হচ্ছে অনেকটা পানির ধারার মতো একটি উৎস থেকে অবিরাম ভাবে প্রবাহ হচ্ছে এমন কিন্তু আমরা ঠিক ভাবে উৎসে কতটুকু পানি আছে জানি না। অর্থাৎ কনসেপচুয়ালি একটি অবিরাম ডাটা প্রবাহ। আমরা এই প্রবাহ থেকে ডাটা পড়তে বা লিখতে পারি। যে কোন স্ট্রিম একটি উৎস বা গন্তব্যস্থলের সাথে সংযুক্ত। উৎস কে বলা হয় ডাটা সোর্স এবং গন্তব্যস্থলকে বলা হয় ডাটা সিংক।



**ইনপুট স্ট্রিম তৈরি**

ছবিতে দেখা যাচ্ছে একটি সোর্স থেকে প্রবাহ আকারে ডাটা ফ্লো হচ্ছে জাভা প্রোগ্রামে। এবং জাভা প্রোগ্রামটি আরেকটি ডাটা ফ্লো তৈরি করছে যা গন্তব্যে পৌছাচ্ছে। তাহলে একটি সোর্স থেকে ডাটা পড়তে হলে আমাদেরকে কয়েকটি ধাপে যেতে হয় - ১. প্রথমে একটি সোর্স নির্ধারণ করতে হবে। সোর্স একটি স্ট্রিং হতে পারে, কিংবা একটি ফাইল অথবা একটি নেটওয়ার্ক কানেকশান। ২. সোর্স এর উপর ভিত্তি করে একটি ইনপুট স্ট্রিম তৈরি করতে হবে। ৩. ইনপুট স্ট্রিম থেকে ডাটা পড়া। সাধারণত একটু লুপ এর মধ্যে ইনপুট স্ট্রিম এর read() মেথড কল করতে হয় , এবং লুপটি ততক্ষণ পর্যন্ত চলে যতক্ষণ পর্যন্ত ডাটা পড়া শেষ না হয়।

**ইনপুট স্ট্রিম থেকে ডাটা পড়া**

স্ট্রিম দুই প্রকার হতে পারে- 1. বাইট স্ট্রিম 2. ক্যারেকটার স্ট্রিম। বাইট স্ট্রিম

বাইট ভিত্তিক আই/ও নিয়ে কাজ করার জন্যে বাইট স্ট্রিম-এ বেশ সমৃদ্ধ ক্লাস আছে। সাধারণত বাইট স্ট্রিম যে কোন টাইপ অবজেক্ট ( যেমন বাইনারী ডাটা) তে ব্যবহার করা যায়। সব বাইট স্ট্রিম এর ক্লাস গুলো InputStream এবং OutputStream এর সাব ক্লাস। যদিও আরও অনেক বাইট স্ট্রিম ক্লাস আছে, কিন্তু যেহেতু এই দুটি ক্লাস সবার উপরে, আমরা শুরুতেই এই দুটি ক্লাস নিয়েই কথা বলবো।

java.io.InputStream এটি একটি অ্যাবস্ট্রাক্ট ক্লাস এবং সকল ইনপুট স্ট্রিম এর সুপার ক্লাস। এতে তিনটি বেসিক মেথড আছে যা কিনা কিভাবে ডাটা স্ট্রিম থেকে পড়তে হয় তা নিয়ে ডিল করে। এছাড়াও স্ট্রিম ক্লোস করা, ফ্লাস করা, এবং কতগুলো বাইট আরও পড়তে হবে ইত্যাদি নিয়ে কিছু মেথড আছে। এগুলো নিয়ে একটি ডিটেইল ব্যাংখ্যা করা যাক। read() মেথড:

public abstract int read() throws IOException

এই মেথডটি ১ বাইট unsigned ডাটা পড়ে এবং এর ইন্টিজার ভ্যালু রিটার্ন করে যা কি না ০ থেকে 255 এর মধ্যে । যদি কোন বাইট না পাওয়া যায় তাহলে এটি -‌1 রিটার্ন করে এবং এতে করে আমরা বুঝতে পারি স্ট্রিম এর ডাটা শেষ হয় গেছে। আমরা একটি উদহারণ দেখি। যেহেতু ইনপুট স্ট্রিম একটি অ্যাবস্ট্রাক্ট ক্লাস এবং এর বেশ কিছু সাব ক্লাস আছে, উদাহরণ দেওয়ার সুবিধার্থে আমরা একটি ফাইল ইনপুট স্ট্রিম ব্যবহার করি যা কিনা কোন একটি লোকেশানে রাখা একটি টেক্সট ফাইল পড়তে পারবে । প্রথমে আমরা একটি টেক্টট ফাইল তৈরি করে কোন একটি লোকেশানে রাখি। সাধারণত প্রজেক্ট এর একটি ফোল্টার তৈরি করে তাতেও রাখা যেতে পারে। এর পর এই ফাইল এ যে কোন একটি স্ট্রিং লিখি। এখানে আমার ফাইল এর নাম input.txt এতে নিচের লাইটি লিখেছি - The quick brown fox jumps over the lazy dog. এবার নিচের কোডটি রান করি।

import java.io.FileInputStream;

import java.io.IOException;

public class InputStreamExample {

public static void main(String[] args) {

FileInputStream in = null;

try {

in = new FileInputStream("input.txt");

int c;

while ((c = in.read()) != -1) {

System.out.print(c + ",");

}

} catch (IOException e) {

System.err.println("Could not read file");

} finally {

if (in != null) {

try {

in.close();

} catch (IOException e1) {

System.err.println("Could close input stream");

}

}

}

}

}

এখানে শুরুতে একটি FileInputStream ক্লাস এর ইনস্টেন্স ক্রিয়েট করো হয়েছে। যেহেতু InputStream একটি abstract ক্লাস, এবং আমাদের ডাটা সোর্স একটি ফাইল, সুতরাং কংক্রিট ক্লাস হিসেবে FileInputStream ব্যবহার করা হয়েছে। এতে আর্গুমেন্ট হিসেবে আমাদের টেক্সট ফাইলটির লোকেশান দেয়া হয়েছে। এখানে এটি রিলেটিভ পাথ। আমাদের ওয়ার্কিং ডিরেকটরী হচ্ছে প্রজেক্ট ডিরেকটরী, যেহেতু ফাইলটি প্রজেক্ট ডিরেক্টরীতেই রাখা আছে। যদি ফাইলটি অন্য ডিরেকটরীতে থাকে সেক্ষেত্রে absolute পাথ দিতে হবে।

তারপর একটা int c ডিক্লেয়ার করা হয়েছে । এরপর একটি হুয়াইল লুপ রয়েছে। এতে প্রতিবার একটি করে বাইট রিড করে c তে এসাইন করা হচ্ছে এবং তা প্রিন্ট আউট করা হচ্চে। এই লুপটি ততক্ষণ পর্যন্ত চলবে যতক্ষণ পর্যন্ত read() মেথডটি -1 রিটার্ন না করে। ফাইল টি পড়া শেষ হয়ে গেলে এটি -1 রিটার্ন করবে। কোডটি একটি ট্রাই ক্যাচ ব্লক এর মধ্যে কারণ আমার জানি যে আই/ও আছে খুব লো-লেভেল থেকে কাজ করে । এর মাঝে কোন একটি সমস্য হতেই পারে এবং তা হলে JVM IOException থ্রু করবে এবং তা যাতে আমরা হ্যান্ডেল করতে পারি। এছাড়াও একটি ফাইনালি ব্লক আছে যেখানে আমরা স্ট্রিমটি বন্ধ করেছি। আমাদের খেয়াল রাখতে হবে যে, যখনি একটি স্ট্রিম এর কাজ শেষ হয়ে যাবে তখনি তা বন্ধ করে দিতে হবে। এটি অনেকটা আমাদের ওয়াশরুমের পানির টেপ এর মতো। কাজ শেষ হলে আমরা অফ করে দিই যাতে করে রিসোর্স নষ্ট না হয়।

এখন উপরের কোডটি যদি রান করি তাহলে কনসোলে আমরা নিচের আউটপুটটি দেখতে পাবো- 84,104,101,32,113,117,105,99,107,32,98,114,111,119,110,32,102,111,120,32,106,117,109,112,115,32,111,118,101,114,32,116,104,101,32,108,97,122,121,32,100,111,103,46,

এর কারণ হচ্ছে read() মেথডটি এক সাথে একটি বাইট পড়ে এবং এর ইন্টিজার রিপ্রেজেন্টেশান রিটার্ন করে। আমরা যদি একে ঠিক আমাদের input.txt এর স্ট্রিং এর মতো করে প্রিণ্ট করতে চাই তাহলে ইন্টিজারকে ক্যারেকটার এ কাস্ট করতে হবে। System.out.print((char)c);

**আউটপুট স্ট্রিম তৈরি**

ছবিতে দেখা যাচ্ছে যে জাভা প্রোগ্রামটি একটি আউটপুট স্ট্রিম ব্যবহার করে একটি ডাটা সিংক ডাটা ট্রান্সফার করছে। আউটপুট স্ট্রিমএর মাধ্যমে প্রোগ্রাম থেকে ডাটা ডাটা সিংকে পাঠাতে হলে কয়েকটি ধাপ-এ যেতে হয়- ১. প্রথমে একটি ডাটা সিংক নির্ধারণ করতে হবে। এটি একটি ফাইল হতে পারে, কিংবা একটি স্ট্রিং অবজেক্ট বা নেটওয়ার্ক কানেকশান। ২. ডাটা সিংক ব্যবহার করে একটি আউটপুট স্ট্রিম অবজেক্ট তৈরি করতে হবে। ৩. এরপর আউটপুট স্ট্রিমটি ফ্লাস করতে হবে। ৪. এবং সবশেষে আউটপুট স্ট্রিমটি ক্লোজ করে দিতে হবে।

**আউটপুট স্ট্রিমে ডাটা রাইট করা**

এবার আমরা চেষ্টা করবো ডাটা কিভাবে ডাটা সিংকে রাইট করা যায় । এক্ষেত্রে ডাটা সিংক হিসেবে একটি ফাইল নিতে পারি। আউটপুট স্ট্রিম হিসেবে নিতে পারি FileOutputStream. OutputStream এর একটি একটি মেথড হচ্ছে write() যা দিয়ে আমরা ডাটা ফাইল এ রাইট করতে পারি। write() মেথড এর কগুলো অভারলোডিং আছে । এর যেকোন একটা ব্যবহার করতে পারি। একটি স্ট্রিং অবজেক্ট থেকে আমরা সহজেই ডাটা বাইট আকারে একটি অ্যারেতে রাখতে পারি।

String text = "Hello";

byte[] textBytes = text.getBytes();

এরপর এই বাইট অ্যারেকে আউটপুট স্ট্রিম এর আরইট মেথডে আর্গুমেন্ট হিসেবে পাস করতে পারি। উদাহরণ-

import java.io.FileOutputStream;

import java.io.IOException;

public class OutputStreamExample {

public static void main(String[] args) {

String destFile = "output.txt";

String data = "Lorem ipsum dolor sit amet," +

" consectetur adipiscing elit. " +

" Suspendisse at placerat ipsum. ";

try {

FileOutputStream fos = new FileOutputStream(destFile);

fos.write(data.getBytes());

fos.flush();

fos.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

এরপর আউটপুট স্ট্রিমটিকে ফ্লাস করতে হয় flush() মেথড ব্যবহার করে। আমাদের উদ্দেশ্য হচ্ছে ডাটা সিংকে ডাটা রাইট করা। এক্ষেত্রে আমরা FileOutputStream এ ডাটা রাইট করছি যা কিনা একটি ফাইল এর অ্যাবস্ট্রাকশন। আউটপুট স্ট্রিম বাইট গুলোকে আপারেটিং সিস্টেম কে দেয় যে কিনা আসলে বাইট গুলো ফাইল এ রাইট করার জন্যে রেসপনসিবল। অপারেটিং সিস্টেম আসলে নির্ধারণ করে কখন বাইট গুলো ফাইল এ রাইট করবে কিন্তু আমাদের আগে সবগুলো বাইট অপারেটিং সিস্টেমকে দিতে হবে। আউটপুট স্ট্রিম এর যেহেতু অ্যাবস্ট্রাক্ট ক্লাস এবং এর অনেক গুলো কনক্রিট আছে, এদের কোন কোন ক্লাস নিজের মাঝে বাইট গুলোর বাফার রেখে দিতে পারে। সেক্ষেত্রে flush() মেথডটি বাফার ক্লিয়ার করে দেবে।

এবং কাজ শেষে আউটপুট স্ট্রিম টিকে ক্লোজ করে দিতে হবে। আউটপুট স্ট্রিম এর আরও বেশ কিছু সাব ক্লাস হলো -

**ক্যারেকটার স্ট্রিম**

ক্যারেক্টার স্ট্রিম গুলো বাইট স্ট্রিম এর মতোই কাজ করে, তবে পার্থক্য শুধু এইটকুই যে এরা ক্যারেকটার নিয়ে কাজ করে। অর্থাৎ এগুলোকে শুধুামাত্র টেক্সট রিড এবং রাইট করার জন্যে লেখা হয়েছে। InputStream এবং OutputStream এর মতো এখানেও দুটি সুপার ক্লাস রয়েছে, যেগুলো হলো - Reader এবং Writer.

ক্যারেক্টার স্ট্রিম সঠিক ভাবে বুঝতে হলে আমাদের আগে জানতে হবে ক্যারেক্টার ইনকোডিং সম্পর্কে। আমরা জানি যে কম্পিউটার মূলত র (raw ) জিরো-ওয়ান নিয়ে কাজ করে। কিন্তু আমরা যখন কোন টেক্সট দেখি তা কিন্তু মোটেও জিরো-ওয়ান বাইনারী ডিজিট নয়, বরং রিয়েল ক্যারেকটার গুলোই দেখি। এই জিরো-ওয়ান বাইনারী ডাটা গুলোকে ইন্টারপ্রেট করার জন্যে এক ধরণের ম্যাপিং থাকে যাকে বলা হয় ক্যারেকটার ইনকোডিং। অনেক ধরণের ক্যারেকটার ইনকোডিং থাকলেও সাধারণত ASCII ও ইউনিকোড-বেইজড ইনকোডিং গুলো নিয়ে আমাদের সমচেয়ে বেশি কাজ করতে হয়। ASCII বা আস্কি - American Standard Code for Information Interchange এর সংক্ষিপ্ত রূপ। এটি একটি ক্যারেকটার ইনকোডিং পদ্ধতি যা ইংরেজী বর্ণ মালা গুলোকে নাম্বারের মাধ্যমে রিপ্রেজন্ট করে। প্রতিটি ইংরেজী বর্ণকে একটি করে নাম্বার (০-১২৭) দেওয়া হয়। এই ইনকোডিং পদ্ধতিতে মাত্র এক বাইট এর দরকার হয়। আস্কি দিয়ে শুধুমাত্র ইংরেজী টেক্সট নিয়ে কাজ করা গেলেও পৃথিবীতে অসংখ্য ভাষা এবং বর্ণমালা রয়েছে। পৃথিবীর সব আধুনিক বর্ণমালা এবং ঐতিহাসিক দলিল গুলো নিয়ে কাজ করার জন্য একটি নতুন পদ্ধতি উদ্ভাবন করা হয়, যার নাম ইউনিকোড। এই ইউনিকোড ইমপ্লিমেন্ট করার জন্যে অনেকগুলো ক্যারেকটার ইনকোডিং স্কিম বা পদ্ধতি রয়েছে, তবে সাধাণত UTF-8, UTF-16 বেশি ব্যবহৃত হয়। UTF-8 ইনকোডিং সিস্টেম এ একটি ক্যারেকটার ১ থেকে ৪ বাইট হতে পারে এবং এটি ওয়েব পেইজ বা ইমেইল ব্যবহৃত হয়। UTF-16 এর ক্ষত্রে তা দই বা ততোধিক বাইট হতে পারে।

অনেক সফটওয়্যার সিস্টেমই UTF ইনকোডিং স্কিম ব্যবহার করে টেক্সট স্টোর করে থাকে। যেহেতু এগুলো একটি ক্যারেকটার রিপ্রেজেন্ট করতে হলে ১ বা একাধিক বাইট দরকার হয়, সেহেতু এগুলো পড়ার সময় যদি আমরা ইনপুটস্ট্রিম ব্যবহার করে একবাইট করে পড়ি, এবং তা char এ কনভার্ট করি, তাহলে আমরা অনেক সময়ই সঠিক ভাবে ডাটা রিড করতে পারবো না। এই সসস্যা দূর করার জন্যে এবং সঠিক ভাবে টেক্সট রিড বা রাইট করার জন্যে Reader/Writer ক্লাস লেখা হয়েছে যা শুধুমাত্র টেক্সট নিয়ে কাজ করে। মনে রাখতে হবে যে, InputStream এর read() মেথড প্রত্যেকবার এক বাইট করে রিটার্ন করে আর Reader ক্লাসের read() মেথড প্রতিবার একটি করে ক্যারেকটার রিটার্ন করে। একটি বাইট এর ভ্যালু ১-২৫৫ পর্যন্ত হতে পারে যেখানে একটি ক্যারেক্টার এর ভ্যালু ০ -৬৫৫৩৫ হতে পারে। তাহলে আমরা সহজ ভাবে বলতে পারি, ইনপুট স্ট্রিম/আউটপুট স্ট্রিম র-বাইনারী ডাটা নিয়ে কাজ করে আর রিডার/রাইটার শুধুমাত্র টেক্সট নিয়ে কাজ করে। এই পার্থক্য ছাড়া ক্যারেক্টার স্ট্রিম নিয়ে কাজ করার সব স্টেপস গুলো ইনপুট/আউটপুট স্ট্রিম এর স্টেপস এর মতো।

**Read using Reader**

import java.io.FileReader;

import java.io.IOException;

import java.io.Reader;

public class ReaderExample {

public static void main(String[] args) {

Reader reader = null;

try {

reader = new FileReader("input.txt");

int c;

while ((c = reader.read()) != -1) {

char ch = (char) c;

System.out.print(ch);

}

} catch (IOException e) {

e.printStackTrace();

}finally{

try {

if (reader!=null) {

reader.close();

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

এতেও ইনপুট স্ট্রিম এর একটা করে বাইট রিড করতে হয় একটি লুপ এর মাঝে।

**Write using Writer**

import java.io.FileWriter;

import java.io.IOException;

import java.io.Writer;

public class WriterExample {

public static void main(String[] args) {

Writer writer;

String text = "Lorem ipsum dolor sit amet,"

+ " consectetur adipiscing elit. "

+ "Suspendisse at placerat ipsum. ";

try {

writer = new FileWriter("output2.txt");

writer.write(text);

writer.flush();

writer.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

```java

রাইটার এর write() মেথড দিয়ে সরাসরি স্ট্রিং রাইট করা যায়।

\*\*System.in, System.out, and System.error

\*\*

এই তিনটি বহুল ব্যবহৃত ডাটা সিংক। তবে সবচেয়ে বেশি ব্যবহৃত হয় মূলত System.out .

System.in একটি ইনপুট স্ট্রিম যা কিনা যে কোন কনসোল প্রোগ্রামের জন্যে কিবোর্ড এর সাথে কানেক্টেড। এটি System ক্লাসের একটি স্যাটিক মেম্বার। এটি যে খুব বেশি ব্যবহৃত হয় তা নয়, তবে আমরা চাইলেই নানা কাজে ব্যবহার করতে পারি। যেমন-

```java

Scanner scanner = new Scanner(System.in);

int a = scanner.nextInt();

double d = scanner.nextDouble();

স্ক্যানার একটি ইউটিলিটি ক্লাস, যার সাহায্যে সহজেই আমরা কিবোর্ড থেকে ইন্টিজার বা ডাবল টাইপ ইনপুট নিতে পারি। স্ক্যানার কনস্ট্রাকর আর্গুমেন্ট হিসেবে একটি ইনপুট স্ট্রিম নেয়। এক্ষেত্রে আমরা System.in টি দিতে পারি যাতে করে এটি সরাসরি কিবোর্ড থেকে ডাটা পড়তে পারে।

System.out হচ্ছে System ক্লাসের একটি স্যাটিক মেম্বার যা কিনা একটি প্রিন্টস্ট্রিম(PrintStream) । এটি যেকোন ডাটা কনসোল এ রাইট করে। এটিও একটি আউটপুট স্ট্রিম তবে এটি ডাটা ফরমেট করে দেখাতে সাহায্য করে। যেমন আমরা যখন কনসোল এ প্রিমিটিভ ডাটা প্রিন্ট করি, প্রিন্ট স্ট্রিম তাদের ফরমেটেড ডাটা গুলো প্রিন্ট করে, এদের বাইট ভ্যালু প্রিন্ট না করে।

System.err ও একটি আউটপুট স্ট্রিম যা কিনা System.out স্ট্রিম এর মতোই কাজ করে , তবে এটি শুধুমাত্র ইরর প্রিন্ট করার জন্যে ব্যবহার করা হয়। কিছু কিছু আইডিই এই ইরর টেক্সট গুলো লাল রং-এ প্রিন্ট করে থাকে।

**রিডিং/রাইটিং প্রিমিটিভ ডাটা**

DataInputStream এবং DataOutputStream ক্লাস দুটি প্রিমিটিভ টাইপ ডাটা কাজ করার জন্যে ব্যবহার করা হয়। এতে বেশ কিছু readxxx() এবং writexxx() মেথড রয়েছে যে গুলো ব্যবহার করে যেকোন ধরণের প্রমিটিভ ডাটা আমরা রিড/রাইট করতে পারি।

উদাহরণ-

import java.io.DataOutputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class WritingPrimitivesExample {

public static void main(String[] args) {

String destFileName = "primitivs.data";

try {

DataOutputStream dos = new DataOutputStream(new FileOutputStream(destFileName));

dos.writeInt(152);

dos.writeDouble(4.56);

dos.writeBoolean(true);

dos.writeLong(Long.MAX\_VALUE);

dos.flush();

dos.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

উদাহরণটিতে DataOutputStream কনস্ট্রাকট আর্গুমেন্ট হিসেবে একটি আউটপুটস্ট্রিম নেয়,এখানে যেহেতু আমরা ফাইল এ রাইট করছি,সে জন্যে FileOutputStream ব্যবহার করা হয়েছে।

import java.io.DataInputStream;

import java.io.FileInputStream;

import java.io.IOException;

public class ReadingPrimitivesExample {

public static void main(String[] args) {

String sourceFile = "primitivs.data";

try {

DataInputStream dis = new DataInputStream(new FileInputStream(sourceFile));

int intValue = dis.readInt();

double doubleValue = dis.readDouble();

boolean booleanValue = dis.readBoolean();

long longValue = dis.readLong();

System.out.println(intValue);

System.out.println(doubleValue);

System.out.println(booleanValue);

System.out.println(longValue);

} catch (IOException e) {

e.printStackTrace();

}

}

}

উদাহরণটিতে DataInputStream কনস্ট্রাকটর আর্গুমেন্ট হিসেবে একটি ইনপুটস্ট্রিম নেয়। যেহেতু আমরা ফাইল থেকে রিড করছি, সেহেতু ইনপুটস্ট্রিম হিসেবে FileInputStream ব্যবহার করা হয়েছে।